NIST SP 800-22: Statistical Test Suite

Testing Random and Pseudorandom Number Generators for Cryptographic Applications

Prof. Michael Mascagni

Department of Computer Science
Department of Mathematics
Department of Scientific Computing
Graduate Program in Molecular Biophysics
Florida State University, Tallahassee, FL 32306 **USA**
E-mail: mascagni@fsu.edu
URL: http://www.cs.fsu.edu/∼mascagni

Outline

Why Test Random Number Generators?

The Core Problem

A deterministic algorithm *cannot* produce truly random output. A sequence that *looks* random may hide exploitable structure.

Where randomness is required in cryptography:

- Symmetric and asymmetric key generation
- Nonces, initialization vectors (IVs), and salts
- Challenge–response protocols and session tokens
- Blinding factors for side-channel protection
- Probabilistic signature schemes (ECDSA, EdDSA)

Fundamental Risk

Weak RNG $\Rightarrow$ predictable secrets $\Rightarrow$ broken cryptographic system — regardless of how strong the cipher is.

Historical Failures: Randomness Gone Wrong

Netscape SSL (1995)

- Keys seeded with `time()` and PID
- Effective entropy: ≈ 20 bits
- Cracked in hours by Goldberg & Wagner

Debian OpenSSL (2008)

- Maintainer removed 2 lines of entropy seeding
- Only $2^{15} = 32{,}768$ distinct keys possible
- Every affected key published online

Sony PS3 ECDSA (2010)

- Constant nonce k reused in every signature
- Private key extracted algebraically from 2 sigs
- Entire console security bypassed

Android Bitcoin (2013)

- Java `SecureRandom` poorly seeded
- ECDSA nonces repeated across transactions
- Private keys recovered; funds stolen

Takeaway

Statistical testing cannot guarantee security, but it *detects structural weaknesses* before deployment.

What is NIST SP 800-22?

Definition

NIST Special Publication 800-22 is a **statistical test suite** published by the National Institute of Standards and Technology for evaluating the quality of random and pseudorandom number generators (RNGs/PRNGs) intended for **cryptographic applications**.

Key facts:

- Published 2001; current revision: **Rev. 1a (2010)**
- Authors: Rukhin, Soto, Nechvatal, Smid, Barker et al. (NIST)
- Contains **15 distinct statistical tests**
- Official reference for FIPS-compliant RNG evaluation
- Free open-source C implementation + full documentation
- URL: `csrc.nist.gov/publications/detail/sp/800-22`

Primary Reference

Rukhin et al., "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications," NIST SP 800-22 Rev. 1a, 2010.

Scope and Limitations

What SP 800-22 Tests

- Bit-level frequency and uniformity
- Serial correlation and run structure
- Spectral (frequency-domain) properties
- Template and pattern matching
- Linear and approximate complexity
- Random walk behavior

What It Does NOT Guarantee

- Cryptographic security
- Next-bit unpredictability
- Forward or backward secrecy
- State recovery resistance
- Passing all 15 tests $\not\Rightarrow$ crypto-safe

Illustrative Example

The Mersenne Twister (MT19937) — Python's `random` module — **passes** all 15 SP 800-22 tests yet is **not** cryptographically secure. Its 19937-bit state can be reconstructed from 624 consecutive 32-bit outputs.

Bottom line: Passing SP 800-22 is **necessary but not sufficient** for cryptographic use.

Hypothesis Testing Review

For every test in SP 800-22:

- H_0 (null): The sequence *is* random (produced by a perfect uniform source)
- H_1 (alternative): The sequence is *not* random

Procedure for each test:

1. Generate or collect a bit sequence of length n
2. Compute a test statistic T from the bits
3. Derive the p-value $= P(T \geq t_{\text{obs}} \mid H_0)$
4. Compare p-value to significance level $\alpha = 0.01$

Decision Rule

- p-value ≥ 0.01 **PASS** — insufficient evidence to reject H_0
- p-value < 0.01 **FAIL** — sequence is non-random at 1% significance

Note: At $\alpha = 0.01$, about 1 in 100 truly random sequences will fail any single test by chance.

Multi-Sequence Testing and Proportion Analysis

NIST recommended practice: Run each test on $m \geq 100$ independent sequences.

Proportion test — how many sequences must pass?

$$\hat{p} = \frac{\text{number of passing sequences}}{m}$$

Acceptable range: $\hat{p} \geq 1 - \alpha - 3\sqrt{\frac{\alpha(1-\alpha)}{m}}$

For $\alpha = 0.01$, $m = 100$: **at least 96 of 100** sequences must pass.

Second-level uniformity test:

- Collect all m p-values from one test
- Apply a chi-square goodness-of-fit test to check uniformity on $[0, 1)$
- Partition $[0, 1)$ into 10 subintervals; expect $m/10$ per bin
- Uniformity p-value ≥ 0.0001 required

Key Insight

A good RNG's p-values should themselves be uniformly distributed. A spike near 0 or 1 signals

Minimum Sequence Lengths

NIST recommended minimum sequence lengths per test:

Test	Min. bits	Notes
Frequency (Monobit)	100	Very short sequences possible
Block Frequency	100	Block size $M \geq 20$
Runs	100	Depends on π estimate
Longest Run of Ones	128	Uses predefined block sizes
Binary Matrix Rank	38,912	Requires full 32×32 matrices
Discrete Fourier Transform	1,000	More bits improve power
Non-overlapping Template	1,000	Per 9-bit template
Overlapping Template	1,000,000	Long sequence critical
Maurer's Universal	387,840	$L = 7, Q = 1280$ default
Linear Complexity	1,000,000	Block length $M = 500$
Serial	1,000	Two consecutive p-values
Approximate Entropy	1,000	Block length $m = 10$
Cumulative Sums	100	Forward and backward
Random Excursions	1,000,000	Needs enough cycles
Random Excursions Variant	1,000,000	18 states tested

Test 1: Frequency (Monobit) Test

Purpose: Check that the number of 1s and 0s is approximately equal.

Statistic: For a sequence $\epsilon_1, \ldots, \epsilon_n$ with $X_i = 2\epsilon_i - 1 \in \{-1, +1\}$:

$$S_n = \sum_{i=1}^{n} X_i, \qquad s_{\text{obs}} = \frac{|S_n|}{\sqrt{n}}, \qquad p\text{-value} = \operatorname{erfc}\left(\frac{s_{\text{obs}}}{\sqrt{2}}\right)$$

Interpretation:

- $S_n \approx 0$ means roughly equal 0s and 1s — ideal
- Large $|S_n|$ indicates bias toward one bit value
- erfc is the complementary error function

Example

Sequence of 100 bits with 55 ones: $S_{100} = 55 - 45 = 10$, $s_{\text{obs}} = 10/\sqrt{100} = 1.0$, $p = \operatorname{erfc}(1/\sqrt{2}) \approx 0.3173$. **PASS.**

Test 2: Frequency Test Within a Block

Purpose: Check that the proportion of 1s within each block of M bits is approximately $1/2$.

Procedure:

1. Divide the n-bit sequence into $N = \lfloor n/M \rfloor$ blocks
2. Compute proportion of 1s in block i: $\pi_i = (\text{ones in block } i)/M$
3. Compute the test statistic:

$$\chi^2_{\text{obs}} = 4M \sum_{i=1}^{N} \left(\pi_i - \tfrac{1}{2}\right)^2$$

$$p\text{-value} = \operatorname{igamc}\left(\frac{N}{2}, \frac{\chi^2_{\text{obs}}}{2}\right)$$

where igamc is the incomplete gamma function complement.

Parameters: $M \geq 20$, $N \geq 100$, and $n \geq M \cdot N$. $M = 128$ is a common choice.

Detects: Local frequency bias that the global monobit test would miss.

Test 3: Runs Test

Purpose: Detect runs of identical bits that are too long or too short relative to a random expectation.

Definition: A **run** is an unbroken sequence of identical bits. $V_n(\text{obs})$ = total number of runs (transitions + 1).

Pre-condition: First check $|\hat{\pi} - 1/2| < 2/\sqrt{n}$ where $\hat{\pi} = (\text{count of 1s})/n$. If this fails, the test automatically fails.

Statistic:

$$p\text{-value} = \operatorname{erfc}\left(\frac{|V_n(\text{obs}) - 2n\hat{\pi}(1-\hat{\pi})|}{2\sqrt{2n}\,\hat{\pi}(1-\hat{\pi})}\right)$$

Detects:

- Too many runs $\Rightarrow$ oscillating sequence (e.g., 01010101...)
- Too few runs $\Rightarrow$ long blocks of same bit

Test 4: Longest Run of Ones in a Block

Purpose: Detect anomalies in the longest run of 1s within each block of M bits.

Block sizes and classification:

M	n min	Categories v_i	χ^2 d.f.
8	128	$\leq 1, 2, 3, \geq 4$	3
128	6,272	$\leq 4, 5, 6, 7, 8, \geq 9$	5
10000	750,000	$\leq 10, \ldots, \geq 16$	5

Statistic:

$$\chi^2(\text{obs}) = \sum_{i=1}^{K+1} \frac{(v_i - N \cdot \pi_i)^2}{N \cdot \pi_i}$$

where π_i are theoretically derived probabilities and v_i = count of blocks with longest run in category i.

Detects: Unusually long or short maximal runs indicating non-randomness in run-length distribution.

Test 5: Binary Matrix Rank Test

Purpose: Check for linear dependence among fixed-length substrings of the sequence by examining the rank of binary matrices.

Procedure:

1. Partition sequence into $M \times Q$ bit matrices
2. Compute the **binary (GF(2)) rank** of each matrix
3. Classify each matrix: full rank M, rank $M-1$, or rank $\leq M-2$
4. NIST uses $M = Q = 32$; need $n \geq 38{,}912$ bits

Expected proportions for random 32×32 binary matrices:

$$P(\text{rank} = 32) \approx 0.2888, \quad P(\text{rank} = 31) \approx 0.5776, \quad P(\text{rank} \leq 30) \approx 0.1336$$

$$\chi^2_{\text{obs}} = \frac{(F_0 - N \cdot 0.2888)^2}{N \cdot 0.2888} + \frac{(F_1 - N \cdot 0.5776)^2}{N \cdot 0.5776} + \frac{(F_2 - N \cdot 0.1336)^2}{N \cdot 0.1336}$$

Detects: Linear dependencies at a global scale — catches generators with poor spectral structure like LCGs.

Test 6: Discrete Fourier Transform (Spectral) Test

Purpose: Detect periodic structure or repeated patterns in the sequence using frequency-domain analysis.

Procedure:

1. Convert bits to ± 1: $X_i = 2\epsilon_i - 1$
2. Compute DFT: $S_k = \sum_{j=0}^{n-1} X_j e^{2\pi ijk/n}$
3. Compute moduli: $M_k = |S_k|$ for $k = 0, \ldots, n/2 - 1$
4. Count peaks exceeding threshold $T = \sqrt{2.995732274 \cdot n}$
5. $N_0 = 0.95 \cdot n/2$ (expected number below threshold)
6. $N_1 =$ observed count below threshold

$$d = \frac{N_1 - N_0}{\sqrt{n \cdot 0.95 \cdot 0.05/4}}, \qquad p\text{-value} = \operatorname{erfc}\left(\frac{|d|}{\sqrt{2}}\right)$$

Detects: Periodic patterns, spectral spikes, frequency-domain biases invisible in the time domain.

Tests 7 & 8: Non-Overlapping and Overlapping Template Tests

Test 7: Non-Overlapping Template Matching

- Search for a specific m-bit pattern (template) in N blocks of M bits
- After each match, restart search m positions ahead (no overlap)
- Count matches per block; test against chi-square with mean μ and variance σ^2
- $\mu = (M - m + 1)/2^m$
- 148 templates tested for $m = 9$
- **Detects:** Specific bit patterns occurring too often or rarely

Test 8: Overlapping Template Matching

- Template: m consecutive 1s (e.g., 1111111111 for $m = 10$)
- Count overlapping occurrences in entire sequence
- Compare observed count distribution to theoretical
- Requires $n \geq 10^6$ bits
- Uses 5 count-categories and χ^2 statistic
- **Detects:** Too many or too few long runs of 1s

Why Both?

Non-overlapping catches specific pattern frequencies. Overlapping catches distributional anomalies in worst-case templates.

Test 9: Maurer's "Universal Statistical" Test

Purpose: Detect whether a sequence can be significantly compressed. A compressible sequence is not random.

Intuition: Based on the distance between matching L-bit blocks. A truly random sequence has maximum entropy and resists compression.

Procedure:

1. Split sequence into blocks of length L (default $L = 7$)
2. Initialization phase: record position of last occurrence of each 2^L possible L-bit pattern using $Q = 1280$ blocks
3. Test phase: for each of the next K blocks, compute $A_i = \log_2(\text{distance to last matching block})$
4. Compute $f_n = \frac{1}{K} \sum_{i=Q+1}^{Q+K} A_i$
5. Compare f_n to expected value μ and variance σ^2 (tabulated in SP 800-22)

$$p\text{-value} = \operatorname{erfc}\left(\frac{|f_n - \mu|}{\sqrt{2}\, c\, \sigma} \right)$$

Test 10: Linear Complexity Test

Purpose: Determine whether the sequence is complex enough to be considered random, as measured by the length of the shortest LFSR that generates it.

Key concept: The **linear complexity** L of a sequence is the length of the shortest linear feedback shift register (LFSR) that produces it. Computed via the **Berlekamp–Massey algorithm**.

Procedure:

1. Divide sequence into N blocks of length $M = 500$ bits
2. Compute L_i for each block using Berlekamp–Massey
3. Compute $T_i = (-1)^M(L_i - M/2) + 2/9$
4. Classify T_i into 7 categories; apply chi-square test

Expected behavior: For random sequences, $L \approx M/2$. Values much smaller indicate predictable, low-complexity structure; values much larger are also suspicious.

Detects: LFSRs and linear generators used directly as RNGs.

Test 11: Serial Test

Purpose: Check that all possible m-bit overlapping patterns appear with equal frequency.

Procedure: For pattern length m (default $m = 16$):

1. Count occurrences of every possible m-bit, $(m-1)$-bit, and $(m-2)$-bit pattern in the cyclic sequence
2. Compute:

$$\psi_m^2 = \frac{2^m}{n} \sum_{k=0}^{2^m-1} \nu_k^2 - n$$

where ν_k = count of pattern k.

$$\delta\psi_m^2 = \psi_m^2 - \psi_{m-1}^2, \qquad \delta^2\psi_m^2 = \psi_m^2 - 2\psi_{m-1}^2 + \psi_{m-2}^2$$

Two p-values are computed from $\delta\psi_m^2$ and $\delta^2\psi_m^2$ using the incomplete gamma function.

Detects: Non-uniform pattern distribution at the m-bit level. Complements the frequency and runs tests at a higher granularity.

Test 12: Approximate Entropy Test

Purpose: Compare the frequency of overlapping blocks of two consecutive lengths (m and $m+1$) to assess regularity.

Approximate Entropy ApEn(m):

$$\phi^{(m)} = \sum_{j=0}^{2^m-1} C_j^{(m)} \log C_j^{(m)}$$

where $C_j^{(m)}$ = proportion of m-bit overlapping patterns equal to pattern j.

$$\text{ApEn}(m) = \phi^{(m)} - \phi^{(m+1)}$$

$$\chi^2_{\text{obs}} = 2n\left(\log 2 - \text{ApEn}(m)\right), \qquad p\text{-value} = \text{igamc}\left(2^{m-1}, \frac{\chi^2_{\text{obs}}}{2}\right)$$

Interpretation:

- High ApEn $\Rightarrow$ high irregularity $\Rightarrow$ more random

Test 13: Cumulative Sums (CUSUM) Test

Purpose: Determine whether the cumulative sum of the ± 1 sequence is too large or too small relative to expectation for a random walk.

Define the random walk:

$$S_k = \sum_{i=1}^{k} X_i, \qquad X_i = 2\epsilon_i - 1 \in \{-1, +1\}$$

Test statistic:

$$z = \max_{1 \le k \le n} |S_k|$$

Two versions: Forward (index $1 \to n$) and Backward (index $n \to 1$) — both are computed and two p-values reported.

$$p\text{-value} = 1 - \sum_{k=\lfloor (-n/z+1)/4 \rfloor}^{\lfloor (n/z-1)/4 \rfloor} \left[\Phi\left(\frac{(4k+1)z}{\sqrt{n}} \right) - \Phi\left(\frac{(4k-1)z}{\sqrt{n}} \right) \right] + \cdots$$

Tests 14 & 15: Random Excursions Tests

Setup: Convert bits to ± 1 walk S_k; identify **cycles** — segments between returns to zero.

Test 14: Random Excursions Test

- For each state $x \in \{-4, -3, -2, -1, +1, +2, +3, +4\}$
- Count how many cycles visit state x exactly k times, $k = 0, 1, 2, 3, 4, \geq 5$
- Compare to theoretical distribution using χ^2
- 8 states $\Rightarrow$ 8 p-values
- Requires ≥ 500 complete cycles

Test 15: Random Excursions Variant

- For each state $x \in \{-9, \ldots, -1, +1, \ldots, +9\}$
- Count *total* visits to state x across all cycles
- Compare to theoretical count $J\sqrt{n}$
- 18 states $\Rightarrow$ 18 p-values
- Tests absolute frequency, not per-cycle distribution

Why Random Excursions?

These detect deviations in the random walk's behavior at specific states — asymmetries invisible to frequency and run tests.

All 15 Tests: Summary Table

#	Test Name	Primary Defect Detected
1	Frequency (Monobit)	Global bit bias
2	Block Frequency	Local block bias
3	Runs	Oscillation or long runs
4	Longest Run of Ones	Run-length distribution
5	Binary Matrix Rank	Linear dependence (global)
6	Discrete Fourier Transform	Periodic/spectral structure
7	Non-Overlapping Template	Specific pattern over-occurrence
8	Overlapping Template	Long-run distribution anomaly
9	Maurer's Universal	Compressibility
10	Linear Complexity	LFSR-like structure
11	Serial	Non-uniform m-bit patterns
12	Approximate Entropy	Regularity/repetitiveness
13	Cumulative Sums	Early-sequence bias
14	Random Excursions	Per-cycle state visit distribution
15	Random Excursions Variant	Total state visit frequency

Running the NIST C Reference Implementation

Build and run:

```
# Clone / download NIST source
git clone https://github.com/terrillmoore/NIST-Statistical-Test-Suite.git
cd sts-2.1.2
make

# Run on a binary file (sequence.bin = raw 0/1 bytes)
./sts -v 1 -m 0 -i 100 -s 1000000 -o results/ -f sequence.bin

# Key flags:
#  -m 0       : test mode (0 = file input)
#  -i 100     : 100 independent sequences
#  -s 1000000: 1,000,000 bits per sequence
#  -o results/: output directory
```

Running the NIST C Reference Implementation (Cont.)

Python alternative (nistrng library):

```python
from nistrng import *
import numpy as np

bits = np.random.randint(0, 2, size=1_000_000, dtype=np.int8)
results = run_all_battery(bits)
for name, result in results:
    status = "PASS" if result.passed else "FAIL"
    print(f"{name:40s}  {status}  p={result.p_value:.6f}")
```

Reading the Output: finalAnalysisReport.txt

```
C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
--- --- --- --- --- --- --- --- --- ---  -------  ----------  ----------------
 6  18  14  14  12   7  10  12   9   8  0.739918     98/100  Frequency
 6  10   9  14  13  10  11  11  11  15  0.739918     96/100  BlockFrequency
10  12   8  14  11  10  12   9  11  13  0.976611    100/100  Runs
 4  12  13  14   6  11  11  12   9   8  0.262249     97/100  LongestRun
 9   6  10  13  12  16  13   8   7   6  0.474986     99/100  Rank
 9   8   8  10  14  12   8  11  11   9  0.974244     98/100  FFT
 *   *   *   *   *   *   *   *   *   *    ------     83/100  NonOverlappingTemplate (B
     =000000001)
 9  10  12   6   7  13  11  10  12  10  0.883171     99/100  OverlappingTemplate
...
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each test with 100 sequences is: 96/100
```

- **C1–C10**: Count of p-values falling in each [0.1) bin — should be ≈ 10 each
- **P-VALUE**: Uniformity chi-square test on the p-value distribution
- **PROPORTION**: Fraction of sequences passing; $\geq 96/100$ required
- ***: Failure flagged — NonOverlappingTemplate for pattern 000000001 fails

Python: Implementing the Monobit Test from Scratch

```python
import numpy as np
from scipy.special import erfc

def monobit_test(bits):
    """NIST SP 800-22 Test 1: Frequency (Monobit) Test."""
    n = len(bits)
    # Map {0,1} -> {-1,+1}
    X = 2 * np.array(bits, dtype=np.int8) - 1
    S_n = np.sum(X)
    s_obs = abs(S_n) / np.sqrt(n)
    p_value = erfc(s_obs / np.sqrt(2))
    return p_value, p_value >= 0.01

# Test on MT19937 (not crypto-secure)
rng = np.random.default_rng(42)
bits_mt = rng.integers(0, 2, size=1_000_000)
p, passed = monobit_test(bits_mt)
print(f"MT19937  -> p={p:.6f}  {'PASS' if passed else 'FAIL'}")
```

Python: Implementing the Monobit Test from Scratch (Cont.)

```python
# Test on ChaCha20 (crypto-secure via secrets module)
import secrets
bits_cc = np.frombuffer(secrets.token_bytes(125000), dtype=np.uint8)
bits_cc = np.unpackbits(bits_cc)[:1_000_000]
p, passed = monobit_test(bits_cc)
print(f"ChaCha20 -> p={p:.6f}  {'PASS' if passed else 'FAIL'}")
```

```
MT19937  -> p=0.412847  PASS
ChaCha20 -> p=0.763291  PASS
# Both pass monobit -- need full suite to distinguish them
```

Failure Diagnosis: What Test Failures Tell You

Test(s) Failing	Likely Root Cause	Generator Type
Frequency (Monobit)	Bit bias; output not 50/50	Poorly designed LCG
Runs + CUSUM	Long alternating or constant runs	Oscillator-based PRNG
Binary Matrix Rank	Linear dependencies	Short-period LCG
DFT (Spectral)	Periodic patterns	LCG or Xorshift low bits
Linear Complexity	Low LFSR complexity	Raw LFSR or linear generator
Non-Overlapping Template	Specific bit pattern bias	Output function flaw
Maurer's Universal	Compressible output	Truncated or biased output
Approx. Entropy	Repetitive structure	Short-period generator
Random Excursions	Walk asymmetry	Sign bias in generator
All tests pass	*No detected weakness*	MT19937, PCG64, ChaCha20

Remember

Passing all tests is *not* a certificate of cryptographic security. It means no *detected* statistical weakness at the tested sequence length.

The NIST RNG Standards Landscape

NIST publishes a family of interrelated standards for randomness:

Publication	Title (abbreviated)	Focus
SP 800-22 Rev. 1a	Statistical Test Suite	Testing RNG output quality
SP 800-90A Rev. 1	DRBG Mechanisms Using Hash/HMAC/AES	CSPRNG algorithms
SP 800-90B	Entropy Sources	Hardware entropy validation
SP 800-90C	DRBG Construction	Full DRBG system design
SP 800-131A Rev. 2	Transitioning Algorithms	Deprecation schedule
FIPS 140-3	Security Requirements for Crypto Modules	Module certification
FIPS 186-5	Digital Signature Standard	RNG requirements for DSA
SP 800-57	Key Management	Key generation randomness

How They Relate

SP 800-22 tests the *output* of an RNG. SP 800-90A defines *which* RNG algorithms are approved. SP 800-90B certifies the *entropy source* feeding the RNG. FIPS 140-3 certifies the complete *cryptographic module*.

NIST SP 800-90A: Approved DRBG Mechanisms

Definition

SP 800-90A specifies **Deterministic Random Bit Generators (DRBGs)** approved for cryptographic use in U.S. federal systems. Rev. 1 (2015) removed the controversial Dual_EC_DRBG.

Currently approved DRBGs (SP 800-90A Rev. 1):

DRBG	Underlying Primitive	Security Strength
Hash_DRBG	SHA-1, SHA-2, SHA-3	112–256 bits
HMAC_DRBG	HMAC-SHA-1/2/3	112–256 bits
CTR_DRBG	AES-128/192/256, 3DES	112–256 bits

Required DRBG operations:

- **Instantiate**: Seed from entropy source; set security strength
- **Generate**: Produce random bits; auto-reseed if needed
- **Reseed**: Inject fresh entropy on demand
- **Uninstantiate**: Zeroize internal state

NIST SP 800-90A: CTR_DRBG (Most Widely Used)

CTR_DRBG is based on **AES in counter mode** and is the most commonly deployed approved DRBG (used in OpenSSL, Linux kernel, Windows CNG).

Internal state:

- V: 128-bit counter value
- K: AES key (128, 192, or 256 bits)
- reseed_counter: counts generate calls since last reseed

Generate operation (simplified):

$$\text{output}_i = \text{AES}_K(V + i), \quad i = 1, 2, \ldots$$

After generating, update K and V using the last two blocks of output.

Security properties:

- **Forward security**: Compromising current state reveals nothing about past output
- **Backtracking resistance**: After reseed, independent of prior state
- Reseed interval: $\leq 2^{48}$ generate requests
- Max bits per request: 2^{19} bits

NIST SP 800-90B: Entropy Sources

Purpose

SP 800-90B specifies requirements and validation procedures for **non-deterministic entropy sources** (hardware RNGs) that seed DRBGs.

Key concepts:

- **Min-entropy:** $H_\infty(X) = -\log_2(\max_i p_i)$ — the conservative entropy measure used for security
- **Full entropy:** Source is considered full-entropy if $H_\infty \geq 0.999$ bits per bit output
- A source with min-entropy h must produce at least n/h bits to collect n bits of entropy

SP 800-90B validation testing:

- Collect ≥ 1 million samples from the source
- Apply 10 entropy estimators (frequency, compression, Markov, etc.)
- The *minimum* estimate is used as the certified entropy rate
- Health tests must be run at startup and continuously during operation

Examples: Intel RDRAND/RDSEED, TPM 2.0 RNG, HSM entropy sources.

FIPS 140-3: Cryptographic Module Validation

Purpose

FIPS 140-3 (2019, supersedes FIPS 140-2) specifies **security requirements for cryptographic modules** used in U.S. federal systems. Aligned with ISO/IEC 19790.

Four security levels:

Level	Physical Security	Typical Use
1	No physical requirements	Software-only modules
2	Tamper-evident coatings	General commercial hardware
3	Tamper-resistant enclosure; zeroize on attack	HSMs, smart cards
4	Complete physical envelope; detect any intrusion	High-security facilities

RNG requirements under FIPS 140-3:

- Must use an SP 800-90A approved DRBG
- Entropy source must meet SP 800-90B requirements
- Continuous health tests required (NIST SP 800-90B Section 4)
- Known-answer tests (KATs) at module initialization

SP 800-131A: Algorithm Transition and Deprecation

Purpose

SP 800-131A Rev. 2 (2019) specifies the **transition schedule** for cryptographic algorithms, including hash functions and RNGs.

Current status for randomness-related algorithms:

Algorithm	Key/Output Size	Status	Notes
SHA-1	160 bits	**Disallowed**	Collision attacks exist
SHA-224/256	224/256	**Approved**	Recommended
SHA-384/512	384/512	**Approved**	High security
SHA-3 family	variable	**Approved**	New standard
3DES	112 bits	**Deprecated**	Through 2023 only
AES-128	128 bits	**Approved**	CTR_DRBG use
Dual_EC_DRBG	—	**Removed**	NSA backdoor suspected

Dual_EC_DRBG

Removed from SP 800-90A in 2014 after Snowden documents confirmed NIST had included a

How SP 800-22 Fits Into the Bigger Picture

The complete NIST randomness validation chain:

1. **Entropy Source** (SP 800-90B)
   Hardware RNG certified for min-entropy; health tests required
2. **DRBG Algorithm** (SP 800-90A)
   CTR_DRBG, Hash_DRBG, or HMAC_DRBG seeded by entropy source
3. **Statistical Validation** (SP 800-22)
   Output of the DRBG tested for statistical quality; all 15 tests applied
4. **Module Certification** (FIPS 140-3)
   Complete cryptographic module validated through CMVP program
5. **Application Use** (SP 800-57, FIPS 186-5)
   Certified randomness used for key generation, signatures, etc.

Key Point

SP 800-22 is one layer of a defense-in-depth approach. Passing it is *necessary* at step 3 but the full chain must hold.

SP 800-22 vs. TestU01 BigCrush vs. Dieharder

Property	**SP 800-22**	**TestU01 BigCrush**	**Dieharder**
Number of tests	15	106 settings	31
Sensitivity	Moderate	Very high	Moderate
Min. bits needed	10^6	$\sim 10^{11}$	$\sim 10^9$
Primary language	C	C	C
Crypto focus	Yes	No	No
FIPS relevance	Yes (required)	No	No
MT19937 result	**Passes**	**Fails 14**	**Borderline**
LCG result	**Fails**	**Fails many**	**Fails**
ChaCha20 result	**Passes**	**Passes**	**Passes**
PCG64 result	**Passes**	**Passes**	**Passes**
Best use	FIPS compliance	RNG design/research	Quick screening

Recommendation

Use SP 800-22 for regulatory compliance. Use TestU01 BigCrush for research-grade generator evaluation. A generator that passes BigCrush but fails SP 800-22 indicates a regulatory gap,

Common Pitfalls When Using SP 800-22

1. **Testing too few sequences**
   Run at least 100 independent sequences; fewer sequences reduce statistical power significantly.
2. **Sequences that are too short**
   Tests like Maurer's Universal and Random Excursions require 10^6 bits minimum. Under-length sequences give meaningless results.
3. **Ignoring the uniformity p-value**
   Many practitioners only check the pass proportion. The second-level uniformity test is equally important.
4. **Passing SP 800-22 $\neq$ cryptographic security**
   MT19937 passes; it is still fully predictable after 624 outputs.
5. **Not testing the actual deployed output**
   Always test the exact output stream from the deployed configuration, not a separately constructed test stream.
6. **Forgetting FIPS requires SP 800-90A**
   SP 800-22 alone does not satisfy FIPS compliance. You need a certified DRBG *and* a certified entropy source.

Summary

NIST SP 800-22 provides 15 tests organized into four themes:

- **Frequency & Runs** (Tests 1–4): Bit-level and run-length uniformity
- **Structure & Patterns** (Tests 5–9): Matrix rank, spectral, and template matching
- **Complexity & Entropy** (Tests 10–12): LFSR complexity, pattern uniformity, compressibility
- **Random Walks** (Tests 13–15): Cumulative sums and excursions

Key takeaways:

- Passing SP 800-22 is *necessary* for FIPS cryptographic use
- It is *not sufficient* — pair with SP 800-90A (DRBG) and SP 800-90B (entropy)
- Always run ≥ 100 sequences of $\geq 10^6$ bits each
- Check both proportion *and* uniformity criteria
- Use TestU01 BigCrush for research-grade RNG evaluation
- For new applications: CTR_DRBG (AES-256) is the safe default

Further Reading and Resources

Primary standards:

- NIST SP 800-22 Rev. 1a (2010) — `csrc.nist.gov/publications/detail/sp/800-22`
- NIST SP 800-90A Rev. 1 (2015) — DRBG algorithms
- NIST SP 800-90B (2018) — Entropy source validation
- FIPS 140-3 (2019) — Cryptographic module validation

Reference implementations and tools:

- NIST C reference: `github.com/terrillmoore/NIST-Statistical-Test-Suite`
- Python wrapper: `pip install nistrng`
- TestU01: `simul.iro.umontreal.ca/testu01`
- Dieharder: `webhome.phy.duke.edu/~rgb/General/dieharder.php`

Papers:

- Rukhin et al., NIST SP 800-22 Rev. 1a, 2010
- L'Ecuyer & Simard, "TestU01," *ACM TOMS* 33(4), 2007
- Shumow & Ferguson, "On the Possibility of a Back Door in the NIST SP800-90 Dual Ec Prng," CRYPTO 2007 Rump Session
- Bernstein et al., "ChaCha20 and Poly1305 for IETF Protocols," RFC 8439, 2018